A Scalable Efficient Robust Adaptive (SERA) Architecture for the Next Generation of Web Service

Jia Wang
Cornell Network Research Group (C/NRG)
Department of Computer Science, Cornell University
Ithaca, NY 14853-7501
jiawang@cs.cornell.edu

Abstract

The World Wide Web can be considered as a large distributed information system that provides access to shared data objects. As one of the most popular applications currently running on the Internet, the World Wide Web is of an exponential growth in size, which results in network congestion and server overloading. Web caching has been recognized as one of the effective schemes to alleviate the service bottleneck and reduce the network traffic, thereby minimize the user access latency. In this paper, we propose a Scalable Efficient Robust Adaptive (SERA) Web caching architecture which accommodates the exponential growth and extreme dynamic environment of the World Wide Web. We developed a piggybacked prefetching/pre-resolving scheme, which uses user access pattern and network environment information. Cooperative consistency control mechanism is employed to further improve the performance. In order to assist the cache resolution, an efficient cache routing scheme is employed. A loose group membership is maintained among nearby proxy caches to make the entire caching system scalable and fault tolerant.

1 Introduction

The World Wide Web (WWW) can be considered as a large distributed information system that provides access to shared data objects. The predicted size of the WWW is shown in Figure 1 [6]. As the WWW continues its exponential growth (the size of static Web pages increases approximately 15% per month), two of the major problems that today's Web users are suffering from are the network congestion and server overloading. The rapid growth of the WWW could be attributed to the fact that at least till now, its usage is quite inexpensive, and accessing information is faster using the WWW than using any other means. Also, the WWW has documents that appeal to a wide range of interests, e.g. news, education, scientific research, sports, entertainment, stock market growth, travel, shopping, weather, maps, etc. Although the Internet backbone capacity increases as 60% per year, the demand for bandwidth is likely to outstrip supply for the foreseeable future as more and more information services are moved onto the Web. If some kind of solution is not undertaken for the problems caused by its rapidly increasing growth, the WWW would become too congested and its entire appeal would eventually be lost.

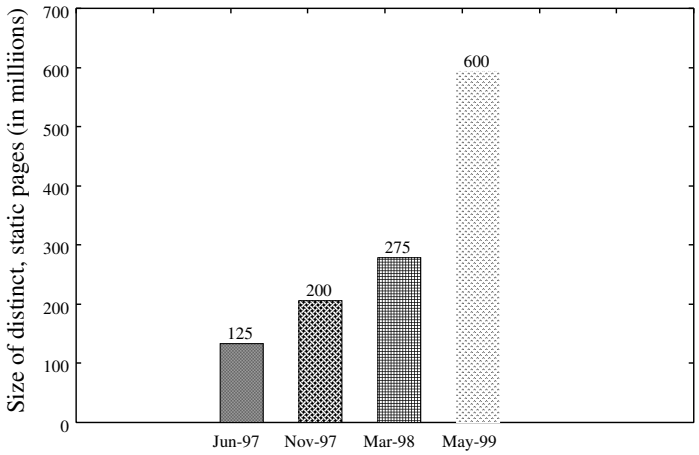

Figure 1: Size of distinct, static Web pages.

Researchers have been working on how to improve Web performance since the early 90's. Caching popular objects at locations close to the clients has been recognized as one of the effective solutions to alleviate Web service bottlenecks, reduce traffic over the Internet and improve the scalability of the WWW system. The idea of using proxy servers [59] to cache arose when they were firstly used to allow accesses to the Internet from users within a firewall (see Figure 2). For security reasons, companies run a special type of HTTP servers called "proxy" on their firewall machines. A proxy server typically processes requests from within a firewall by forwarding them to the remote servers, intercepting the responses, and sending the replies back to the clients. Since the same proxy servers are typically shared by all clients inside of the firewall, naturally this leads to the question of the effectiveness of using these proxies to cache documents. Clients within the same firewall usually belong to the same organization and likely share common interests. They would probably access the same set of documents and each client tends to browse back and forth within a short period of time. Therefore on the proxy server a previously requested and cached document would likely result in future hits. Web caching at proxy server can not only save network bandwidth but also lower access latency for the clients.

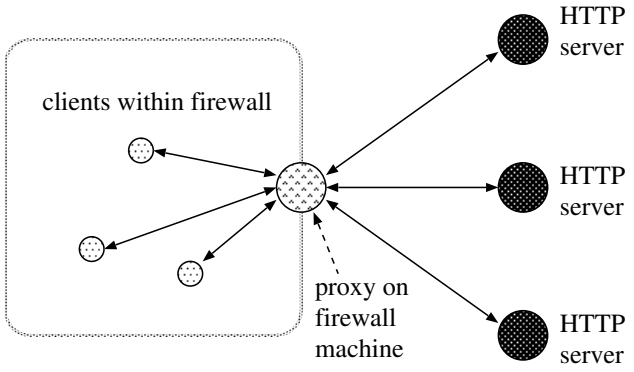

Figure 2: A proxy running on the firewall machine to service clients inside the subnet.

Documents can be cached on the clients, the proxies, and the servers. The effects of Web caching are two-fold. First, it has been shown that caching documents can improve Web performance significantly [16] [30] [51].

There are several advantages of using Web caching.

1. Web caching reduces bandwidth consumption, thereby decreases network traffic and lessens network congestion.
2. Web caching reduces access latency due to two reasons:
   (a) Frequently accessed documents are fetched from nearby proxy caches instead of remote data servers, the transmission delay is minimized.
   (b) Because of the reduction in network traffic, those documents not cached can also be retrieved relatively faster than without caching due to less congestion along the path and less workload at the server.
3. Web caching reduces the workload of the remote Web server by disseminating data among the proxy caches over the wide area network.
4. If the remote server is not available due to the remote server's crash or network partitioning, the client can obtain a cached copy at the proxy. Thus, the robustness of the Web service is enhanced.
5. A side effect of Web caching is that it provides us a chance to analyze an organization's usage patterns.

Furthermore, as suggested by [49] [54], a group of caches cooperating with each other in terms of serving each other's requests and making storage decisions result in a powerful paradigm to improve cache effectiveness. However, it's worth to note that there are several disadvantages of using a caching system in Web services.

1. The main disadvantage is that a client might be looking at stale data due to the lack of proper proxy updating.
2. The access latency may increase in the case of a cache miss due to the extra proxy processing. Hence, cache hit rate should be maximized and the cost of a cache miss should be minimized when designing a caching system.
3. A single proxy cache is always a bottleneck. A limit has to be set for the number of clients a proxy can serve. An efficiency lower bound (i.e. the proxy system is ought to be at least as efficient as using direct contact with the remote servers) should also be enforced.
4. A single proxy is a single point of failure.
5. Using a proxy cache will reduce the hits on the original remote server which might disappoint a lot of information providers, since they cannot maintain a true log of the hits to their pages. Hence, they might decide not to allow their documents to be cacheable.

A lot of research work have been done to study the effect of Web caching and maximize its benefits. There are several subtle issues on employing a caching system to facilitate Web services which need to be studied and solved (e.g. proxy location, cache routing, dynamic data caching, etc). A naive caching system may actually

degrade Web performance drastically and introduce instabilities into network [35]. Intelligent and careful design is crucial to improve the quality of Web service. In the following sections, we will then discuss a line of research that addresses these problems and try to solve them.

The remainder of this paper is organized as follows. Section 2 outlines the elements of a World Wide Web caching system and Section 3 describes some desirable characteristics. Section 4 gives a brief survey of previous work on schemes to improve Web performance and the research frontier in this area is given in Section 5. An overview of our Web caching architecture is described in Section 6. Section 7 describes the advantages of our approach. Finally, We conclude our work and address future work in Section 8.

2 Elements of a World Wide Web caching system

A generic model of Web caching system is shown in Figure 3. In such system, documents can be cached at the clients, the proxies, and the servers. A client always requests page from its local proxy if it doesn't have a valid copy of such page in its own browser's cache. Upon receiving a request from client, the proxy first checks to see if it has the requested page. If so, it returns the page to the client. If it doesn't have the requested page in its cache, it sends a request to its cooperative proxies or the server. Upon receiving a request from another proxy, a proxy checks if it has the requested page. If so, it returns the page to the requesting proxy. If not, the proxy may further forward the request to other proxies or the server. If none of the cooperative proxies has such page, the requested page is fetched from the server.

In order to make a WWW caching system work, the following problems need to be solved properly:

- How are the cache proxies organized, hierarchically, distributed, or hybrid? (caching system architecture)
- Where to place a cache proxy in order to achieve optimal performance? (proxy placement)
- What can be cached in the caching system, data, connection, or computation? (caching contents)
- How do proxies cooperate with each other? (proxy cooperation)
- What kind of data/information can be shared among cooperated proxies? (data sharing)
- How does a proxy decide where to fetch a page requested by a client? (cache resolution/routing)
- How does a proxy decide what and when to prefetch from Web server or other proxies to reduce access latency in the future? (prefetching)
- How does a proxy manage which page to be stored in its cache and which page to be removed from its cache? (cache placement and replacement)
- How does a proxy maintain data consistency? (cache coherency)
- How does proxy/server distributed data objects? (data disemination)
- How is the control information (e.g. URL routing information) distributed among proxies? (control information distribution)

- How to deal with data which is not cacheable? (dynamic data caching)

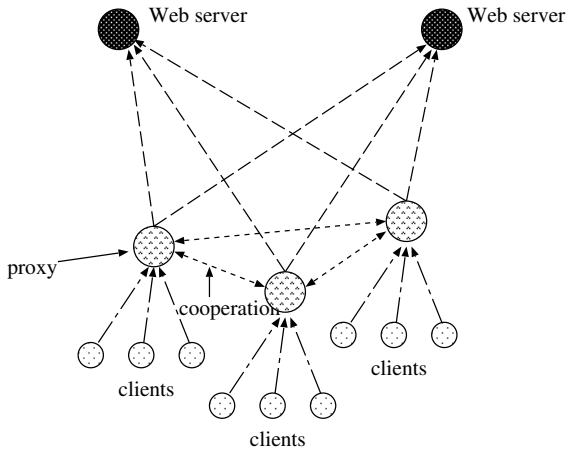

Figure 3: A generic WWW caching system.

These questions must be addressed in every reasonable caching system. Depending upon the choices made in answering each question, a variety of schemes have been proposed. They will be discussed in Section 4.

3 Desirable properties of WWW caching system

Besides the obvious goal of Web caching system, we would like a Web caching system to have a number of properties. They are:

- Fast access
- Robustness
- Transparency
- Scalability
- Efficiency
- Adaptivity
- Stability
- Load balancing
- Ability to deal with heterogeneity
- Simplicity

We discuss these in turn.

3.1 Fast access

From users' point of view, access latency is an important measurement of the quality of Web service. A desirable caching system should aim at reducing Web access latency. In particular, it should provide user a lower latency on average than those without employing a caching system.

3.2 Robustness

From users' prospect, the robustness means availability, which is another important measurement of quality of Web service. Users desire to have Web service available whenever they want. The robustness has three aspects. First, it's desirable that a few proxies crash wouldn't tear the entire system down. The caching system should eliminate the single point failure as much as possible. Second, the caching system should fall back gracefully in case of failures. Third, the caching system would be design in such a way that it's easy to recover from a failure.

3.3 Transparency

A Web caching system should be transparent for the user, the only results user should notice are faster response and higher availability.

3.4 Scalability

We have seen an explosive growth in network size and density in last decades and is facing a more rapid increasing growth in near future. The key to success in such an environment is the scalability. We would like a caching scheme to scale well along the increasing size and density of network. This requires all protocols employed in the caching system to be as lightweight as possible.

3.5 Efficiency

There are two aspects to efficiency. First, how much overhead does the Web caching system impose on network? We would like a caching system to impose a minimal additional burden on the network. This includes both control packets and extra data packets incurred by using a caching system. Second, the caching system shouldn't adopt any scheme which leads to under-utilization of critical resources in network.

3.6 Adaptivity

It's desirable to make the caching system adapt to the dynamic changing of the user demand and the network environment. The adaptivity involves several aspects: cache management, cache routing, proxy placement, etc. This is essential to achieve optimal performance.

3.7 Stability

The schemes used in Web caching system shouldn't introduce instabilities into the network. For example, naive cache routing based on dynamic network information will result in oscillation. Such an oscillation is not desirable

since the network is under-utilization and the variance of the access latency to a proxy or server would be very high.

3.8 Load balancing

It's desirable that the caching scheme distributes the load evenly through the entire network. A single proxy/server shouldn't be a bottleneck (or hot spot) and thereby degrades the performance of a portion of the network or even slow down the entire service system.

3.9 Ability to deal with heterogeneity

As networks grow in scale and coverage, they span a range of hardware and software architectures. The Web caching scheme need adapt to a range of network architectures.

3.10 Simplicity

Simplicity is always an asset. Simpler schemes are easier to implement and likely to be accepted as international standards. We would like an ideal Web caching mechanism to be simple to deploy.

4 Previous work

Having described the attributes of an ideal Web caching system, we now survey some schemes described in the literature and point out their inadequacies.

4.1 HTTP

The Hyper Text Transfer Protocol (HTTP) is an application-level protocol for distributed collaborative, hypermedia information systems. HTTP has been in use by the World Wide Web global information interactive since 1990. Although HTTP can not be considered as Web caching schemes, understanding the features which HTTP provides us and making use of them in designing Web caching schemes are beneficial.

Currently two standard versions of HTTP are present: HTTP/1.0 [45] and HTTP/1.1 [46]. In spite of its popularity, it's widely understood that HTTP/1.0 has numerous flaws. HTTP/1.1 differentiates itself from HTTP/1.0 by improvement in several areas such as the extensibility, caching, bandwidth optimization, network connection management, message transmission, Internet address conservation, error notification, security, integrity, authentication, content negotiation, etc [52]. However, in absence of adequate understanding and experiments with HTTP/1.1, it's hard to conclude how these changes affect the Web performance in practice.

4.2 Examples of recent Web caching systems

Several recent caching systems are briefly described here. They are:

1. Harvest cache

2. Cachemesh system
3. Summary cache
4. Adaptive Web caching
5. Access driven cache

4.2.1 Harvest cache

Harvest cache servers [17] are organized in a hierarchy with parents and siblings and they co-operate using a cache resolution protocol called Internet Cache Protocol (ICP) [92]. When a cache server receives a request for an object that misses, it sends a request to all its siblings and parents via remote procedure call. The object will be retrieved from the site with lowest latency. There are some problems with the Harvest cache.

1. *Communication overhead and latency.* The Harvest cache uses a simple request/reply protocol for cache resolution. For each local cache miss, the cache server has to send a query message to all its neighbors and parents, and receive a reply back from each of them. The request/reply interaction also introduces some latency during the resolution.
2. *Search overhead.* The search for a cached object in Harvest cache is done in a distributed fashion. For each query, all neighbors and parents have to search their caches in parallel, no matter whether the results is a cache hit or a cache miss.
3. *Cache placement.* In Harvest cache, there is no explicit cache placement policy. An object may be cached at any of the cache servers. This lack of structure in cache placement may force all cache servers to be searched in order to find out whether an object is cached or not.

4.2.2 Cachemesh system

Cachemesh is a distributed cache system for Web [87]. It uses co-operative cache placement and cache routing to achieve close and effective co-operation among a group of cache servers. In a Cachemesh system, cache servers establish a cache routing table among them, and each cache server becomes the designed cache server for a number of Web sites. User requests are then forwarded to the proper cache server according to the cache routing table. Cachemesh can significantly reduce latency, search overhead, and space utilization in a co-operating cache system. However, only a preliminary plan in Cachemesh system is given, detailed schemes (e.g. how to compute cache routing table) have not been exploited yet. This idea is a motivation for our SERA caching architecture.

4.2.3 Summary cache

Summary cache [34] is a scalable wide area cache sharing protocol. Each proxy improves cache sharing by keeping a compact summary of the cache directory of every participating proxy. When a client request misses in the local cache, the proxy checks these summaries for potential hits. If a hit occurs, the proxy sends request to the relevant proxies to fetch the Web page. Otherwise, the proxy sends request directly to the Web server. The proxy

updates the copies of summary stored at other proxies until a certain percentage of its cached documents are not reflected in other proxies. Bloom filters [5] are used to keep each individual summary small. Experiments have shown that Summary cache reduces the number of inter-cache protocol messages, bandwidth consumption, and protocol CPU overhead significantly while maintaining almost the same cache hit ratio as ICP (Internet Caching Protocol) [92]. However, when the number of proxies is large, the network traffic for updating cache contents could also be large. In particular, a proxy may not be interested in every piece of summaries it gets from others. Thus, Summary cache may have poor scalability.

4.2.4 Adaptive Web caching

Adaptive Web caching [69] is an adaptive, scalable, and robust caching system. Cache servers are self-organized and form into a tight mesh of overlapping multicast groups and adapt as necessary to changing conditions. This mesh of overlapping groups forms a scalable, implicit hierarchy that is used to efficiently diffuse popular Web content towards the demand. Adaptive Web caches exchange their entire content state with other members of their cache groups to eliminate the delay and unnecessary use of resources of explicit cache probing. For each request, a cache first determines whether a copy of the requested Web page is cached by one of its group members. If not, the request will be forwarded to another cache in the Web caching infrastructure which is significantly more likely to have the data using a URL routing table maintained at each Web cache. The information in the URL routing table is learned from the source-based and content-based entries from other Web caches. However, this work is still in its preliminary stage and we don't know what they did exactly in order to achieve their goal.

4.2.5 Access driven cache

Access driven Web caching [96] is a scheme using proxy profiles and information groups which are based on Web page access patterns to reduce the average number of messages among proxies for updating the cache status (comparing to Summary cache [34]) while maintaining a high cache hit ratio. Association rule discovery is used to find and summarize the most prevalent access patterns to Web pages from trace data. Such information is then used to partition the Web pages into clusters. All proxies which frequently access some page in the same Web page cluster form an information group. When a host wants to access a Web page, it sends a request to the local proxy. If the page is not cached in the local proxy, a request is sent to the "nearest" site (either a proxy in the same information group or the Web server) to fetch the page. When proxy cache content changes, only proxies in the same information group are notified. Furthermore, a new cache replacement algorithm is proposed, which takes into account not only the local reference pattern, but also whether a page is cached on some other proxy in the same information group. This scheme is shown to greatly reduce the number of messages and overhead on individual proxies while maintaining a high cache hit rate. However, when number of proxies is large, establishing and maintaining information group based on user access patterns are expensive.

4.3 Caching architectures

The performance of a Web cache system depends on the size of its client community; the bigger is the user community, the higher is the probability that a cached document (previously requested) will soon be requested

again. Caches sharing mutual trust may assist each other to increase the hit rate. A caching architecture should provide the paradigm for proxies to cooperate efficiently with each other.

4.3.1 Hierarchical caching architecture

One approach to coordinate caches in the same system is to set up a caching hierarchy. With hierarchical caching, caches are placed at multiple levels of the network. For the sake of simplicity, we assume that there are four levels of caches: bottom, institutional, regional, and national levels [80]. At the bottom level of the hierarchy there are the client/browser caches. When a request is not satisfied by the client cache, the request is redirected to the institutional cache. If the document is not found at the institutional level, the request is then forwarded to the regional level cache which in turn forwards unsatisfied requests to the national level cache. If the document is not found at any cache level, the national level cache contacts directly the original server. When the document is found, either at a cache or at the original server, it travels down the hierarchy, leaving a copy at each of the intermediate caches along its path. Further requests for the same document travel up the caching hierarchy until the document is hit at some cache level.

Hierarchical Web caching was first proposed in the Harvest project [17]. Other examples of hierarchical caching are Adaptive Web caching [69], Access Driven cache [96], etc. A hierarchical architecture is more bandwidth efficient, particularly when some cooperating cache servers do not have high-speed connectivity. In such a structure, popular Web pages can be efficiently diffused towards the demand. However, there are several problems associated with a caching hierarchy [80] [84]:

1. To set up such a hierarchy, cache servers often need to be placed at the key access points in the network. This often requires significant coordination among participating cache servers.
2. Every hierarchy level may introduce additional delays.
3. High level caches may become bottlenecks and have long queueing delays.
4. Multiple copies of the same document are stored at different cache levels.

4.3.2 Distributed caching architecture

Recently, a number of researchers have proposed the setup of a totally distributed caching scheme, where there are only caches at the bottom level. In distributed Web caching systems [72] [84], there are no other intermediate cache levels than the institutional caches, which serve each others' misses. In order to decide from which institutional cache to retrieve a miss document, all institutional caches keep meta-data information about the content of every other institutional cache. To make the distribution of the meta-data information more efficient and scalable, a hierarchical distribution mechanism can be employed. However, the hierarchy is only used to distribute directory information about the location of the documents, not actual document copies. With distributed caching, most of the traffic flows through low network levels, which are less congested and no additional disk space is required at intermediate network levels. In addition, distributed caching allows better load sharing and are more fault tolerant. Nevertheless, a large-scale deployment of distributed caching may encounter several problems such as high connection times, higher bandwidth usage, administrative issues, etc. [80].

There are several approaches to the distributed caching. The Harvest group designed the Internet Cache Protocol (ICP) [92], which supports discovery and retrieval of documents from neighboring caches as well as parent caches. Another approach to distributed caching is the Cache Array Routing protocol (CARP) [86], which divides the URL-space among an array of loosely coupled caches and lets each cache store only the documents whose URL are hashed to it. Provey and Harrison also proposed a distributed Internet cache [72]. In their scheme upper level caches are replaced by directory servers which contain location hints about the documents kept at every cache. A hierarchical meta-data-hierarchy is used to make the distribution of these location hints more efficient and scalable. Tewari et al. proposed a similar approach to implement a fully distributed Internet caching system where location hints are replicated locally at the institutional caches [84]. In the central directory approach (CRISP) [41], a central mapping service ties together a certain number of caches. In Cachemesh system [87], cache servers establish a cache routing table among them, and each cache server becomes the designed server for a number of Web sites. User requests are then forwarded to the proper cache server according to the cache routing table. In Summary Cache [34], Cache Digest [79], and the Relais project [77], caches inter-exchange messages indicating their content and keep local directories to facilitate finding documents in other caches.

4.3.3 Hybrid caching architecture

In a hybrid scheme, caches may cooperate with other caches at the same level or at a higher level using distributed caching. ICP [92] is a typical example. The document is fetched from a parent/neighbor cache that has the lowest RTT. Rabinovich et al. [76] proposed to limit the cooperation between neighbor caches to avoid obtaining documents from distant or slower caches, which could have been retrieved directly from the origin server at a lower cost.

4.3.4 Performance of caching architectures

The main performance measure is the expected latency to retrieve a Web document. It's debatable that which caching architecture can achieve the optimal performance. A recent research work [80] shows that hierarchical caching has shorter connection times than distributed caching, and hence, placing additional copies at intermediate levels reduces the retrieval latency for small documents. It's also shown that distributed caching has shorter transmission times and higher bandwidth usage than hierarchical caching. A "well configured" hybrid scheme can combine the advantages of both hierarchical and distributed caching, reducing the connection time as well as the transmission time. However, these results are based on the assumption that the underlying network topology is a full balanced d-ary tree. It's still not clear that which architecture is optimal for Web caching.

4.4 Cache resolution/routing

Scalability and deployment concerns have led most designers of Web caching infrastructures to schemes based on deploying a large number of Web caches scattered over the Internet. The main challenge in such approaches is how to quickly locate a cache containing the desired document. While this problem is similar to the general problem of network routing, it cannot be solved in the same way. Conventional routing protocols scale because of the route aggregation made possible by hierarchical addressing. However, since documents with the same

URL prefixes or server address prefixes will not necessarily be delivered to the same clients, there is no necessary location among their cache locations. With no way to aggregate routes, the cache routing tables would be unmanageably large. In addition, they have to be updated frequently. Out-of-date cache routing information leads to cache misses. In order to minimize the cost of a cache miss, an ideal cache routing algorithm should route requests to the next proxy which is believed to contain the desired document and along (or close to) the path from the client towards the Web server.

The common approach is to grow a caching distribution tree away from each popular server towards sources of high demand and do cache resolution either via *cache routing table* or via *hash functions*. This works well for requests for very popular documents, because these documents will propagate out to many caches, and so will be found quickly. For less popular documents, the search may follow a long and circuitous path of numerous failed checks. The impact of this is substantial since the hit rate on Web caches is typically less than 50%, indicating a large number of documents of have only low to moderate popularity [2].

4.4.1 Cache routing table

Malpani et al. [68] work around this problem by making a group of caches function as one. A user's request for a page is directed to an arbitrary cache. If the page is stored there, it's returned to the user. Otherwise, the cache forwards the requests to all other caches via IP multicast. If the page is cached nowhere, the request is forwarded to the home site of the page. The main disadvantages is that as the number of participating caches grows, the number of messages between caches can become unmanageable.

Harvest cache system [17] organizes caches in a hierarchy and uses a cache resolution protocol called Internet Cache Protocol (ICP) [92]. Requests for Web documents are forwarded up the hierarchy in search of a cached copy. In attempt to keep from overloading caches at the root, caches query their siblings before passing requests upwards. This approach lengthens the duration of each check. As the hierarchy grows to accommodate more users and more documents, there is an advise impact on the numerous requests for objects that are not every popular.

Adaptive Web Caching [69] uses a mesh of caches in which distribution trees for each server are built. The caches in the mesh are organized into overlapping multicast groups through which a request travels in search of a cached document. This scheme benefits from constructing different distribution trees for different servers (so no root node will be overloaded) and being robust and self-configuring. For less popular objects, queries travel through many caches, and each check requires a query to and responses from a group of machines. The authors suggest dealing with this problem by limiting the number of caches a request will access, but this may greatly reduce the hit rate for everything other than the most popular pages.

Provey and Harrison [72] construct a manually configured hierarchy that must be traversed by all requests. Their scheme is promising in the way that it reduces load on top-level caches by only keeping location pointers in the hierarchy, but it suffers from the same inefficiencies for less popular documents.

Wang [87] describes a preliminary plan in Cachemesh system to put cache routing tables in caches to specify, for each page or server, where to look next if the local cache does not hold the document. A default route for some documents would help to keep table size reasonable.

To reduce the time needed to find relatively unpopular, but cached, documents and the latency of searching

for documents that are not cached, Legedza and Guttag [61] integrate the routing of requests with the datagram routing services already provided by the network layer. However, this approach has deployment problem since it involves a lot of changes and extensions to network routers, packet headers and endpoint protocol processing.

4.4.2 Hashing function

The Cache Array Routing Protocol (CARP) [86] allows for "queryless" distributed caching by using a hash function based upon the "array membership list" and URL to provide the exact cache location of an object, or where it will be cached upon downloading from the Internet. When one proxy server is added or removed, $1/n$ URLs need to be reassigned and the new hash function need to be distributed among proxies, where n is the number of proxy servers.

In Summary cache [34], each proxy keeps a summary of the URLs of cached documents at each participating proxy and checks these summaries for potential hits before sending any queries. To reduce the overhead, the summaries are stored as a Bloom filter [5] and updated only periodically. Experiments have shown that Summary cache reduces the number of inter-cache protocol messages, bandwidth consumption, and protocol CPU overhead significantly while maintaining almost the same cache hit ratio as ICP (Internet Caching Protocol) [92]. However, when the number of proxies is large, the network traffic for updating cache contents could also be large. Thus, Summary cache may have poor scalability.

Karger et al. [50] describe a theoretically based technique for constructing per-server distribution trees with good load balancing properties using a special kind of hashing called *consistent hashing*. A consistent hash function is one which only needs minimal changes as the range of the function changes. Consistent hashing technique is designed to relieving hot spots on the WWW and is still under development. Additionally, it may solve the reassignment problem present in CARP [86]. However, since server load is not taken into account when determining tree size, the trees for small and/or lightly loaded servers may be too big, resulting in many useless application-level checks. Also, since trees are random, the client-server paths through them are likely to be much longer than the regular routing path from client to server.

4.5 Prefetching

Although Web performance is improved by caching documents at proxies, the benefit from this technique is limited [29] [51]. Previous research has shown that the maximum cache hit rate can be achieved by any caching algorithm is usually no more than 40% to 50%. In other words, regardless of the caching scheme in use, one out of two documents can not be found in cache [2]. One way to further increase the cache hit rate is to anticipate future document requests and preload or prefetch these documents in a local cache.

Prefetching can be applied in three ways in the Web contexts:

1. Between browser clients and Web servers.
2. Between proxies and Web servers.
3. Between browser clients and proxies.

4.5.1 Between browser clients and Web servers

Early studies focus on the prefetching schemes between browser clients and Web servers. Padmanabhan and Mogul [74] analyze the latency reduction and network traffic of prefetching using Web server traces and trace-driven simulation. The prediction algorithm they used is based on the Prediction-by-Partial-Matching (PPM) data compressor with prefix depth of 1. The study shows that prefetching from Web servers to individual clients can reduce client latency by 45% at the expense of doubling the network traffic. Bestavros and Cunha [8] present a model for speculative dissemination of World Wide Web documents. The work shows that reference patterns observed at a Web server can be used as an effective source of information to drive prefetching, and reaches similar results as [73]. Cunha and Jaccoud use [20] a collection of Web client traces and study how effectively a user's future Web accesses can be predicted from his or her past Web accesses. They show that a number of models work well and can be used in prefetching. Crovella and Barford [15] analyzed the network effects of prefetching and shows that prefetching can reduce access latency at the cost of increasing network traffic and increasing network traffic burstiness (and thereby increasing network delays). They proposed a rate-controlled prefetching scheme to minimize the negative network effect by minimizing the transmission rate of prefetched documents. However, these early studies do not consider or model caching proxies and hence fail to answer the question about performance of prefetching completely.

4.5.2 Between proxies and Web servers

After proxies have been used to assist Web access, research interest has been shifted to investigating prefetching techniques between proxies and Web servers. Kroeger et al. [51] investigate the performance limits of prefetching between Web servers and proxies, and show that combining perfect caching and perfect prefetching at the proxies can at least reduce the client latency by 60% for high bandwidth clients. Markatos and Chronaki [66] propose that Web servers regularly push their most popular documents to Web proxies, which then push those documents to the clients. They evaluate the performance of the strategy using several Web server traces and find that this technique can anticipate more than 40% of a client's request. The technique requires cooperation from the Web servers. The study does not evaluate client latency reduction from the technique. Cohen et al. [21] also investigate similar techniques. Wcol [26] is a proxy software that prefetches documents, links, and embedded images. The proxy, however, does not push the documents to the client. Gwertzman and Seltzer [42] discuss a technique called Geographical Push-Caching where a Web server selectively sends its documents to the caches that are closest to its clients. The focus of the study is on deriving reasonably accurate network topology information and using the information to select caches.

4.5.3 Between browser clients and proxies

Prefetching can also be done between browser clients and proxies. Loon and Bharghavan [60] proposed a design and an implementation of a proxy system that performs the prefetching, image filtering, and hoarding for mobile clients. Fan et al. [36] proposed an approach to reduce latency by prefetching between caching proxies and browsers. The approach relies on the proxy to predict which cached documents a user might reference next (based on PPM data compressor), and takes advantage of idle time between user requests to either push or pull

the documents to the user. Simulation results show that prefetching combined with large browser cache and delta-compression can reduce client latency up to 23.4%.

4.5.4 Summary

The first two approaches run the risk of increasing wide area network traffic, while the last one only affects the traffic over the modems or the LANs. All of these approaches attempt to prefetch either documents that are considered as popular at servers or documents that are predicted to be accessed by user in the near future based on the access pattern. Network environment information is not taken into account to determine which document and when to be prefetched. I believe that, a network-aware prefetching scheme will minimize the negative network effects caused by prefetching while still achieving similar performance benefit (i.e. reducing access latency and increasing hit rate) as shown in previous work.

4.6 Cache placement/replacement

The key aspect of the effectiveness of proxy caches is a document placement/replacement algorithm that can yield high hit rate. While cache placement has not been well studied, a number of cache replacement algorithms have been proposed in recent studies, which attempt to minimize various cost metrics, such as hit rate, byte hit rate, average latency, and total cost. They can be classified into the following three categories as suggested in [3].

1. Traditional replacement policies and its direct extensions:
   - *Least Recently Used* (LRU) evicts the object which was requested the least recently.
   - *Lease Frequently used* (LFU) evicts the object which is accessed least frequently.
   - *Pitkow/Recker* [91] evicts objects in LRU order, except if all objects are accessed within the same day, in which case the largest one is removed.
2. Key-based replacement policies: (i.e. the replacement policies in this category evict objects based upon a primary key. Ties are broken based on secondary key, tertiary key, etc.)
   - *Size* [91] evicts the largest object.
   - *LRU-MIN* [2] biased in favor of smaller objects. If there are any objects in the cache which have size being at least S, LRU-MIN evicts the least recently used such object from the cache. If there are no objects with size being at least S, then LRU-MIN starts evicting objects in LRU order of size being at least $S/2$. That is, the object who has the largest log(size) and is the least recently used object among all objects with the same log(size) will be evicted first.
   - *LRU-Threshold* [2] is the same as LRU, but objects larger than a certain threshold size are never cached.
   - *Hyper-G* [91] is a refinement of LFU, break ties using the recency of last use and size.
   - *Lowest Latency First* [90] minimizes average latency by evicting the document with the lowest download latency first.

3. Cost-based replacement policies: (i.e. the replacement policies in this category employ a potential cost function derived from different factors such as time since last access, entry time of the object in the cache, transfer time cost, object expiration time and so on.)

   - *GreedyDual-Size* (GD-Size) associates a cost with each object and evicts object with the lowest cost/size.
   - *Hybrid* [90] associates a utility function with each object and evicts the one has the least utility to reduce the total latency.
   - *Lowest Relative Value* [64] evicts the object with the lowest utility value.
   - *Least Normalized Cost Replacement* (LCN-R) [83] employs a rational function of the access frequency, the transfer time cost and the size.
   - *Bolot/Hoschka* [11] employs a weighted rational function of transfer time cost, the size, and the time last access.
   - *Size-Adjusted LRU* (SLRU) [3] orders the object by ratio of cost to size and choose objects with the best cost-to-size ratio.
   - *Server-assisted* scheme [22] models the value of caching an object in terms of its fetching cost, size, next request time, and cache prices during the time period between requests. It evicts the object of the least value.
   - *Hierarchical GreedyDual* (Hierarchical GD) [49] does object placement and replacement cooperatively in a hierarchy.

To sum up, a great deal of effort has been made to maximize the hit rate. However, the performance of replacement policies depends highly on traffic characteristics of WWW accesses. No known policy can outperform others for all Web access patterns.

4.7 Cache coherency

Caches provide lower access latency with a side effect: every cache sometimes provide users with *stale* pages - pages which are out of date with respect to their master copies on the Web servers where the pages originated [31]. Every Web cache must update pages in its cache so that it can give users pages which are as fresh as possible. Caching and the problem of cache coherency on the World Wide Web are similar to the problems of caching in distributed file systems. However, the Web is different than a distributed file system in its access patterns, its larger scale, and its single point of updates for Web objects [43].

4.7.1 HTTP commands that assist Web proxies in maintaining cache coherence

Before dealing with the cache coherence mechanisms, we first give a brief overview about the commands that HTTP [45] provides to assist Web proxies in maintaining cache coherence.

1. *HTTP GET*. Retrieves a document given its URL.

2. *Conditional GET*. HTTP *GET* combined with the header *IF-Modified-Since*. *date* can be used by proxies to ask a remote server to return a copy only if it has been modified.

3. *Pragma:no-cache*. This header appended to *GET* can indicate that the object is to be reloaded from the server irrespective of whether it has been modified or not. Most browsers like Netscape offer a *Reload* button which uses this header to retrieve the original copy.

4. *Last-Modified:date*. Returned with every *GET* message and indicates the last time the page was modified.

5. *Date:date*. The last time the object was considered to be fresh; this is different from the *Last-Modified* header. Netscape, one of the most popular browsers, does not provide a mechanism for displaying the value of a page's *Date* header in the *Document Info* window.

The above commands have been used by the clients via Web proxies. If a remote server receives a *Conditional GET* request but does not support it, it just sends the entire document. However, Loutonen and Altis reported in [59] that at least all major HTTP servers already support the *Conditional GET* header.

4.7.2 Cache coherence mechanisms

Current cache coherency schemes providing two types of consistency. *Strong cache consistency* and *weak cache consistency* have been proposed and investigated for caches on the World Wide Web.

1. Strong cache consistency

   (a) *Client validation*. This approach is also called *polling-every-time*. The proxy treats cached resources as potentially out-of-date on each access and sends an *If-Modified-Since* header with each access of the resources. This approach can lead to many 304 responses (HTTP response code for "*Not Modified*") by server if the resource does not actually change.

   (b) *Server invalidation*. Upon detecting a resource change, the server sends invalidation messages to all clients that have recently accessed and potentially cached the resource [24]. This approach requires a server to keep track of lists of clients to use for invalidating cached copies of changed resources and can become unwieldy for a server when the number of clients is large. In addition, the lists themselves can become out-of-date causing the server to send invalidation messages to clients who are no longer caching the resource.

2. Weak cache consistency

   (a) *Adaptive TTL*. The adaptive TTL (also called Alex protocol [14]) handles the problem by adjusting a document's time-to-live based on observations of its lifetime. Adaptive TTL takes advantage of the fact that file lifetime distribution tends to be bimodal; if a file has not been modified for a long time, it tends to stay unchanged. Thus, the time-to-live attribute to a document is assigned to be a percentage of the document's current "age", which is the current time minus the last modified time of the document. Studies [14] [42] have shown that adaptive TTL can keep the probability of stale documents within reasonable bounds ($< 5\%$). Most proxy servers (e.g. CERN *httpd* [59] [88]) use

this mechanism. The Harvest cache [17] mainly uses this approach to maintain cache consistency, with the percentage set to 50%. However, there are several problems with this expiration-based coherence [31]. First, user must wait for expiration checks to occur even though they are tolerant to the staleness of the requested page. Second, if a user is not satisfied with the staleness of a returned document, they have no choice but to use a *Pragma:No-Cache* request to load the entire document from its home site. Third, the mechanism provides no strong guarantee towards document staleness. Forth, users can not specify the degree of staleness they are willing to tolerate. Finally, when the user aborts a document load, caches often abort a document load as well.

(b) *Piggyback Invalidation*. Krishnamurthy et al. propose piggyback invalidation mechanisms to improve the effectiveness of the cache coherency [21] [55] [56] [57]. Three invalidation mechanisms are proposed. The Piggyback Cache Validation (PCV) [55] capitalizes on requests sent from the proxy cache to the server to improve coherency. In the simplest case, whenever a proxy cache has a reason to communicate with a server it piggybacks a list of cached, but potentially stale, resources from that server for validation. The basic idea of the Piggyback Server Invalidation (PSI) mechanism [56] is for servers to piggyback on a reply to a proxy, the list of resources that have changed since the last access by this proxy. The proxy invalidates cached entries on the list and can extend the lifetime of entries not on the list. They also proposed a hybrid approach which combines the PSI and the PCV techniques to achieve the best overall performance [57]. The choice of the mechanism depends on the time since the proxy last requested invalidation for the volume [21]. If the time is small, then the PSI mechanism is used, while for longer gaps the PCV mechanism is used to explicitly validate cache contents. The rationale is that for short gaps, the number of invalidations sent with PSI is relatively small, but as the time grows longer the overhead for sending invalidation will be larger than the overhead for requesting validations.

To sum up, all above data coherence schemes validate/invalidate objects by obtaining corresponding metadata of the requested objects from the Web server. If the Web server is far away, it may take a long time to get proper information. We believe, data coherence can be conducted by obtaining such information from cooperative proxies, and thereby reduce the total access latency.

4.8 Caching contents

Cache proxy has been recognized as an effective mechanism to improve Web performance. A proxy may serve in three roles: *data cache*, *connection cache*, and *computation cache*. A recent study has shown that caching Web pages at proxy reduces the user access latency 3% - 5% comparing to the no-proxy scheme. In presence of P-HTTP, a proxy can be used as a connection cache. By using persistent connections between clients and proxy and between proxy and Web server, the total latency improvements grow substantially (i.e. 20% - 40%) [16] [35]. However, no cache scheme has been proposed to make full utilization of connection caching.

Computation caching can be viewed as the Web server replicates and migrates some of its services to the proxies to alleviate the server bottleneck. One application of such computation caching is dynamic data caching. The motivation is that, in presence of current caching schemes, the hit ratio at proxy is at most 50%, which is limited by the fact that a significant percentage of Web pages is dynamically generated and thereby is not

cacheable. Computation caching can be used to improve the performance to retrieve dynamic data by caching dynamic data at proxies and migrating a small piece of computation to proxies [19] [27] [63] to generate or maintain the cached data.

4.9 User access pattern prediction

Proxy's policies for managing cached resources (i.e. prefetching, coherence, placement and replacement) and TCP connections rely on the assumptions about client access pattern. To improve the information exchanges between Web servers and proxies, a variety of techniques have been proposed to predict the future requests. One set of approaches are to group resources that likely to be accessed together based on the likelihood that pairs of resources are accessed together, server file system structure, etc [23] [96]. Others are using Prediction by Partial Match (PPM) model to predict which page is likely to be accessed in near future [36] [73] [74]. However, user access pattern is quite hard to predict due to the locality of Web requests. Current schemes are still relative expensive and inaccurate. A more efficient, and yet effective, access pattern prediction algorithm is important to assist proxy management.

4.10 Load balancing

Many of us have experienced the hot spot phenomenon in the context of Web. Hot spots occur any time a large number of clients wish to simultaneously access data or get some services from a single server. If the site is not provisioned to deal with all of these clients simultaneously, service may be degraded or lost. Several approaches to overcoming the hot spots have been proposed. Most use some kind of replication strategy to store copies of hot pages/services throughout the Internet; this spreads the work of serving a hot page/service across several servers (i.e. proxies) [17] [44] [68] [75]. However, load balancing should be done cooperatively with cache routing, prefetching, cache placement/replacement, and proxy placement. Purely replicating data throughout the Internet is not effective enough to accommodate the dynamic changing in the user demands and network environment.

4.11 Proxy placement

The placement of proxies is also important to achieve optimal Web performance. The desirable properties of such proxy placement policies are self-organizing, efficient routing, efficient placement, load balancing, stable behavior, etc. However, little study has been done to address this issue. Li et al. [62] attempted to solve it based on the assumptions that the underlying network topologies are minimum spanning tree and modeled it as a dynamic programming problem. Due to the unrealistic assumption, their solution doesn't give us much insightful suggestion to completely solve the proxy placement problem.

4.12 Dynamic data caching

As we mentioned before, the benefit of current Web caching schemes is limited by the fact that only a fraction of web data is cacheable. Non-cacheable data (i.e. personalized data, authenticated data, server dynamically generated data, etc.) is of a significant percentage of the total data. For example, measurement results show that 30% of user requests contain cookies [16] [35]. How to make more data cacheable and how to reduce the latency

to access non-cacheable data have become crucial problems in order to improve Web performance. Current approaches can be classified into two categories: *active cache* and *server accelerator*.

Active cache [27] supports caching of dynamic documents at Web proxies by allowing servers to supply cache applets to be attached to documents and requiring proxies to invoke cache applets upon cache hitting to finish the necessary processing without contacting the server. It's shown that Active cache scheme can result in significant network bandwidth saving at the expense of CPU cost. However, due to the significant CPU overhead, the user access latency is much larger than that without caching dynamic objects.

Web server accelerator [63] resides in front of one or more Web servers to speed up user accesses. It provides an API which allows application programs to explicitly add, delete, and update cached data. The API allows the accelerator to cache dynamic as well as static data. Invalidating and updating cached data is facilitated by the Data Update Propagation (DUP) algorithm which maintains data dependence information between cached data and underlying data in a graph [19]. However, this scheme is aiming at alleviate the bottleneck of the Web servers, it doesn't help to reduce the network traffic. In case of a network congestion exists, the user may still experience large access latency.

4.13 Web traffic characteristics

Understanding the nature of the workloads and system demands created by users of the World Wide Web is crucial to properly designing and provisioning Web services. The effectiveness of caching schemes relies on the presence of temporal locality in Web reference streams and on the use of appropriate cache management policies that appropriate for Web workloads. A number of measurements have been done to exploit the access properties at clients, proxies, and servers [1] [7] [9] [29] [30].

4.14 Conclusion

From the survey above we draw the following conclusions:

- While great efforts have been made to improve the Web performance, most of them are aiming at reduce user access latencies. The availability is an important aspect of Web performance too. But this has not been well studied.

- The bottleneck of Web access may exist at clients, network, and servers. The request latency can be broken down into browser, server, and network components. Which component contributes most to the total access latency is still an open issue. Previous work mainly focused on how to alleviate the network bottleneck in order to reduce the access latency. This is important because the research work should target on minimizing the major components of the total access latency.

- A number of the current Web caching schemes organize co-operated proxies in a hierarchy while some researchers have proposed distributed caching system in order to enhance the scalability and robustness of the entire system. However, it's not clear that which structure, hierarchical or distributed, can achieve optimal Web performance for general network topologies in practice.

- The current cache proxy management schemes are based on static information such as the cache size, user access pattern, and etc. The network information such as topology, available bandwidth, and propagation latency are not integrated very well in page prefetching, placement and replacement policies, proxy placement, and load balancing. Performance benefit from Web caching is limited without take the network environment information into account on designing the cache policies. How to obtain proper network information and use it to facilitate managing proxy caches is still under study. Moreover, while making the caching system adaptive to network environment, it shouldn't introduce instabilities into the network. Further more, proxy cooperation has been introduced into cache management, however, current cache management schemes have not taken full advantages of such cooperation. For example, proxies still need to contact servers for data coherence purpose.

- The cache resolution/routing is important in order to reduce access latency. Where and how to request a page is a hard decision problem in the Web caching system because proxies don't have the complete and accurate information on which proxy has the requested page and a cache miss is expensive. There is a dilemma here. If all proxies request page directly from the server, then the server becomes a bottleneck and overall network traffic increases. These will result in large access latency. If the proxy fetch the requested page from a nearby proxy, there is a risk that a cache miss may occur due to the inaccurate information, which may result in an even larger latency. The client can only benefit from proxy caching only if the requested page can be fetched from proxies which is faster than doing so from the original Web server.

- The placement of proxies is important to achieve load balancing, route efficiency, fault tolerance, etc. However, where to place a proxy is a hard decision problem.

- It is recognized that connection caching is more effective than data caching to improve the Web performance. However, no scheme has been proposed to take advantage of connection caching.

- The benefit of current Web caching schemes are limited by the fraction of total data accessed by users which are cacheable. Non-cacheable data (i.e. personalized data, authenticated data, server dynamically generated data, etc.) is of a significant percentage of the total. How to make more data cacheable and how to reduce the latency to access non-cacheable data have become critical problems in order to improve Web performance.

- In the current Web caching scheme, co-operated proxies trust each other. However, this may not be true in reality. The performance of Web access would be degraded drastically if a proxy intends to send incorrect information to others. The schemes used to solve such problem in local distributed system is too heavy-weight and not easily extended to the wide area network. How to detect such a malicious proxy and prevent it from crashing down the entire system is still under study.

- Reliable multicast is used to exchange/distributed the URL routing information and other control information among proxies/servers. However, as more and more proxies and servers running on the wide area network, employing reliable multicast is too heavy-weight. How to distribute control information reliably and efficiently among participating proxies/servers is an open problem. A more light-weight protocol need to be used to exchange control information among proxies/servers.

In summary, the above issues have not been completely addressed in the existing literature. A naive caching system can actually degrade Web performance drastically and introduce instabilities into network [35]. Intelligent and careful design is necessary to improve the quality of Web service.

5 Research direction

The research frontier in Web performance improvement lies in developing efficient, scalable, robust, adaptive, stable Web caching scheme that can be easily deployed in current network. Existing schemes fail to solve the above problems. We propose a Scalable Efficient Robust Adaptive (SERA) Web caching architecture which accommodates the exponential growth and extreme dynamic environment of the World Wide Web. In SERA caching system, Web pages are grouped by their Web sites (or subdirectories of Web site). In addition, Web sites are grouped into *communities* according to the servers location, user access pattern and hyperlink topology. Such aggregation of information greatly facilitate cache resolution, cache validation and object prefetching. Since the requests of Web pages of a particular Web site are aggregated in the greatest degree, connection caching, which has been proved to be more important to improve Web performance than purely data caching [16] [35], can be applied easily. Furthermore, each proxy cache may have a local private cache, which is not visible to other proxy caches, and a public cache, which can be shared by nearby proxies. The Web pages frequently accessed by local clients are cached in the local private cache. The proxy public cache stores popular Web pages of one or more Web site communities, which is controlled by the cooperative network-aware cache placement/replacement algorithms. Piggybacked prefetching/pre-resolving is also employed based on the user access pattern and network environment information. Cooperative consistency control mechanism is employed to further improve the performance. In order to assist the cache resolution, an efficient cache routing scheme is employed. A loose group membership is maintained among nearby proxy caches to make the entire caching system scalable and fault tolerant.

6 SERA Web caching architecture

To accommodate the exponential growth and extreme dynamic environment of the World Wide Web, we propose a Scalable Efficient Robust Adaptive (SERA) Web caching architecture. Our goal is to build a co-operative Web caching architecture on which the end-to-end Web performance can be substantially improved in the presence of current HTTP protocols (i.e. HTTP/1.0 and HTTP/1.1) and Web proxy infrastructure. The general architecture of SERA Web caching system is shown in Figure 4.

6.1 Caching system structure

The performance of Web caches depends on the size of client community connected to it; the more people use it, the higher the probability that a given document has already been requested and is presented in the cache. Caches cooperate to increase the probability to hit a document. A caching architecture should provide a paradigm that assists proxies cooperate efficiently with each other.

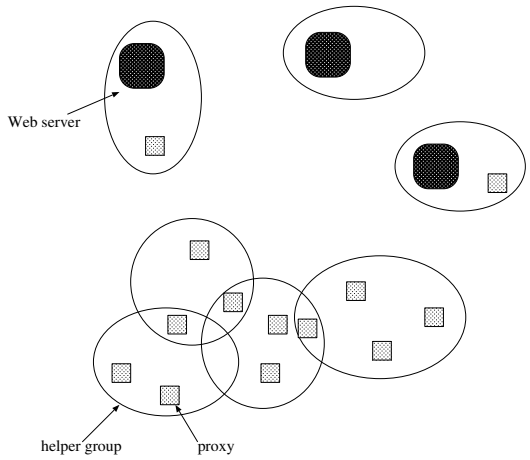

Figure 4: The general architecture of SERA Web caching system.

In SERA Web caching system, nearby cache servers (i.e. Web servers and proxy caches) are organized into a mesh of overlapping groups, which are called *helper groups*, based on the communication latency and contents stored at cache servers. The enhanced Web performance is achieved by proper cooperation among cache servers. The cache servers inside one helper group exchange information (i.e. Web site community information, cache management information, and cache routing information) with each other via customized *pbcast* (probabilistic broadcast) [12] protocol. A loose group membership is then maintained among nearby proxy caches to make the entire caching system scalable and fault tolerant.

The entire structure of the caching system can be set up either hierarchically, distributed, or hybrid. Hierarchical and distributed structures are two ends of the spectrum, hybrid approach is in the middle. A hierarchical architecture is more bandwidth efficient, particularly when some co-operating cache servers do not have high speed connectivity. In such a structure, popular Web pages can be efficiently diffused towards the demand. However, to set up such a hierarchy, cache servers often need to be placed at the key access points in the network. This often requires significant coordination among participated cache servers. In a distributed structure, all cache servers operate at the same level. This allows better load sharing, less traffic concentration and more fault tolerant. However, these benefits come at a price of less bandwidth efficiency. In a hybrid structure, several proxies cooperate at each level using distributed caching. It's shown that this structure can outperform both hierarchical and distributed structures for a full balanced d-ary tree network topologies [80]. In the design of our SERA caching system, although the system is set up to form a hybrid topology, we plan to study hierarchical, distributed, and hybrid cache server topologies.

6.2 Caching contents

In SERA Web caching system, proxies act in three roles: data caching, connection caching, and computation caching. Traditional caching systems only cache data at proxies to reduce the user access latency. Recent research work has shown that connection caching by maintaining persistent connections between proxies and Web servers are even more effective to optimize Web performance than pure data caching. By aggregating user requests inside helper group, the Web performance will benefit more from caching connections between cache

servers.

How to make more data cacheable has become a crucial problem in optimizing Web performance. Caching computation at proxies is necessary to cache dynamic data, and thereby reduce the cost to access them. This is one of the applications of computation caching (or called service replication/migration). In SERA system, we plan to develop efficient and effective scheme to deploy computation caching at proxies.

6.3 Network-aware cooperative cache management

Cache management is one of the key aspects of the effectiveness of Web caching system. It includes cache placement and replacement, prefetching, cache coherence, etc. An end-to-end approach to improving Web performance using server volumes and proxy profiles has been proposed in [21]. The server groups related resources into volumes based on the file system directory structure and the access pattern. The customized volume information is piggybacked to requesting proxy to improve the effectiveness of a variety of proxy policies such as cache coherency, prefetching, cache replacement, and so on. The proxy generated filter, which indicates the type of information of interest to the proxy, are used to tailor the piggyback information. Experimental results show that probability-based volumes can achieve higher prediction rate with lower piggyback size than the directory-based structure. Further reductions in processing and memory overhead are possible by limiting the calculation of probability implications to pairs of resources that have the same directory prefix at the expense of missing associations between resources in different directories. However, conditional probability might not always be the best measurement of correlation of page accesses. Furthermore, cache sharing and proxy co-operating is not addressed in this work.

In SERA caching system, we'll employ a network-aware cooperative piggybacked cache management scheme which is similar to the piggybacked scheme proposed in [21] to further improve Web performance. The new scheme is extended in following ways. First, the volume (we call it Web site community) construction is based on the access pattern, file system structure as well as the hyperlink topologies to capture the correlation between different Web sites. Second, the proxy generated filter, which is used to tailor the piggybacked information, indicates the type of information of interests to the entire helper group (not just the proxy itself) on its responsible Web sites. Third, to further improve the benefit from piggybacked information, we introduce learning process into the piggybacked scheme by asking proxies to piggybacking an evaluation. Finally, to further improve the Web performance, the piggybacked information from the server can be shared by the proxies within the same helper group which will reduce the need for that information to traverse the long path between nearby proxies and the Web server. Our new scheme mainly consists of the following elements:

- cost-aware cache placement and replacement scheme
- Web site community construction scheme
- network-aware piggybacked prefetching and pre-resolving scheme
- cooperative consistency control scheme
- enhanced piggybacked scheme

We describe them in turn.

6.3.1 Web sites community construction

In SERA caching system, Web pages are grouped by their Web sites (or subdirectories of Web site). In addition, Web sites are grouped into *communities* according to the user access pattern. In previous research work, user access pattern is modeled by file system structure, conditional probabilities [21] [96], which lose correlation among Web servers and on which the volume (or Web site community) construction based still involves too many information as the size of Web grows rapidly. Recent theory advances can bring even more improvements for Web performance. The hyperlink topology of Web pages has a strong impact on user access patterns. User usually follows the hyperlinks to visit a sequence of Web pages. Also, related Web pages have great possibility that hyperlinks point to each other. This forms hyperlinked communities [39] [40] [48]. Such hyperlink topologies information can be extracted from parsing HTML. The Web sites community can be constructed based on a combined scheme of an aggregated hyperlinked Web site community and probability-based access pattern. This can be used to guide the prefetching, cache coherency, cache replacement, and cache sharing. Unlike the scheme based on file system directory structure [21], the hyperlink and probability-based scheme eliminates the drawback of missing associations between resources in different directories.

The construction of Web sites communities can be done either hierarchically at the cache servers of each level or at one or more selected cache servers, which are called *community managers*. one example of such community managers is the Web server.

6.3.2 Cache placement and replacement

A key aspect of the effectiveness of proxy caches is a document placement/replacement algorithm that can yield high hit rate, reduce access latency and maintain load balancing. In SERA system, a cooperative cache placement and replacement algorithm is developed to accommodate the user demands and maintain fault tolerance. Each proxy cache may have a local private cache, which is not visible to other proxy caches, and a public cache, which can be shared by nearby proxies (Figure 5). The Web pages frequently accessed by local clients of a proxy are cached in its local private cache. The proxy public cache stores popular Web pages of one or more Web site communities which will be shared among all the proxies of one helper group. This can be considered as partial replicas of corresponding Web sites, which only contain the popular documents accessed by local clients. The cache placement of the public caches should be performed cooperatively based on the following rules:

1. The frequently accessed Web sites of the local users should be cached at some proxies in the cache server helper group.
2. A proxy should cache those Web sites which are heavily accessed by its local users.
3. A proxy should cache those Web sites which are not cached by other proxies in the same cache server helper group.
4. The Web sites cached at the public proxy caches change adaptively to the user access pattern.
5. The load balancing should be maintained among proxies.

A cached Web site may be removed from the proxy public cache under the following circumstances:

1. The workload is too high or the space is exhausted, the cache server may choose to remove some of its cached Web sites.

2. The Web site is rarely accessed by users, there is little gain for cache server to cache it.

3. The Web site belongs to a new Web sites community, which is cached at another cache server, due to the change of user access pattern or the hyperlink topology, it will be gradually migrated to other cache server. This requires proper coordination among cache servers.

To sum up, a cooperative cache placement and replacement algorithm will be developed to accommodate the user demands and maintain fault tolerance.

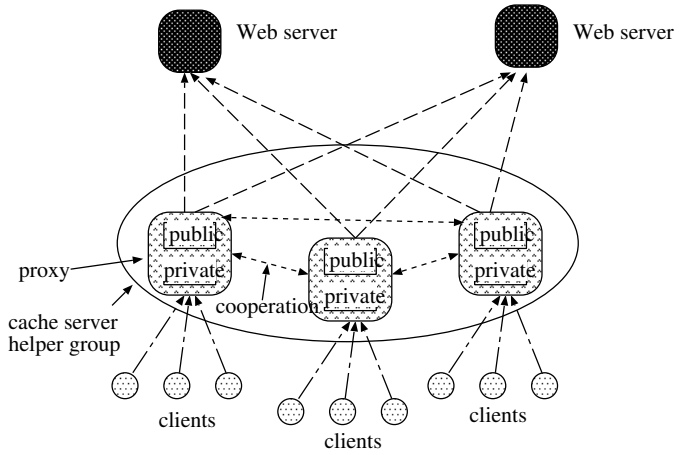

Figure 5: The cache server helper group.

6.3.3 Network-aware piggybacked prefetching and pre-resolving

Previous work attempt to prefetch either documents that are considered as popular pages at servers or documents that are predicted to be accessed by user in the near future based on the access pattern. Network environment information is not taken into account to determine which document and when to be prefetched. In our SERA system, we propose a network-aware piggybacked prefetching and pre-resolving scheme which will minimize the negative network effects caused by prefetching while still achieving similar performance benefit (i.e. reducing access latency and increasing hit rate) as shown in previous work.

In SERA, our network-aware piggybacked prefetching scheme makes use of network environment information obtained from TCP to determine when, what, and how to prefetch. An efficient network performance discovery architecture called SPAND (Shared PAssive Network Discovery) has been proposed in [82], which will provide us a useful tool to obtain desired information of network performance such as effective bandwidth, loss rate, etc. These information will be fed into the prefetching process to make the prefetching decision based not only on the predicted access pattern, but also on the network environment.

In addition, prefetching is combined with pre-resolving scheme to further improve the performance. That is, in some cases, instead of prefetching entire page from sever or other proxies to its cache, the proxy pre-resolves

the location of the desired page and prefetches a portion of it. The rest part of the page will be fetched when user requests it. The performance will benefit most from the combined scheme in the case of the cost of prefetching a page is larger than that of pre-resolving the page.

Finally, whether or not to use multicast instead of using unicast to preload/push pages to proxies if more than one of them are interested in these pages is also an interesting problem to study. The efficient of data disemination depends on the size of data which will be distributed, number of proxies interested in the data, network topology, link loss rate, etc.

6.3.4 Cooperative consistency control

Cache consistency control has two functionality. First, it reduces the probabilities by which user gets stale pages. Current data coherence schemes validate/invalidate objects by obtaining corresponding metadata of the requested objects from the Web server. In the case of an invalid (potential stale) object, it still pay the full cost of contacting the remote server. If the Web server is far away, it may take a long time to get proper information.

Second, it maintains a consistent view of the meta information (e.g. what are currently cached at proxies) among cooperative proxies.

A cooperative consistency control scheme is deployed in SERA system. It consists of two aspects. First, page validation/invalidation can be done by querying nearby proxies. Second, meta information can be disseminated via customized gossip protocol.

6.3.5 Enhanced piggybacked scheme

In SERA system, cache resource and TCP connections management is guided by information piggybacked from server. The new piggybacked scheme is enhanced in following ways. First, to further improve the benefit from piggybacked information, a learning process is introduced into the piggybacked mechanism by asking proxy piggyback an evaluation. Second, the information piggybacked from server are shared among cooperative proxies. Third, the proxy generated filter, which is used to tailor the piggybacked information, indicates the type of information of interests to the entire helper group (not just the proxy itself) on its responsible Web sites. Finally, a gossip protocol is integrated into piggyback mechanism to reduce the cost of gossiping.

6.4 Cache resolution/routing

The cache routing problem can be further divided into several subproblems.

- Service auto-discovery/advertisement;
- Cache routing table computation;
- Routing requests.

6.4.1 Service auto-discovery/advertisement

We can consider requesting a particular page to be requesting a kind of service from the cache server. There are two ways for proxies to learn the current available services and its corresponding cost for receiving such services:

service auto-discovery and *service advertisement*. Both schemes will be examined in SERA system. The *pbcast* and BGP-like protocols will be used to facilitate such discovery/advertisement both inside and across domains, respectively. It may be helpful to set up a special name server in each domain.

6.4.2 Cache routing table computation

By grouping Web pages into Web site community, the cache routing information can be aggregated, and thereby reduce the routing table size substantially. Two tables need to be maintained at each proxy: *Web sites community table* and *cache routing table*. The Web sites community table contains information for those Web sites communities which are cached at proxies with the same cache server helper group. The entry in the Web sites community table has the following format:

<community ID, list of Web site URL>

The entry of the cache routing table is of the format:

<community ID, list IP address of cache server>

For the Web site community for which it is not responsible, the proxy stores the IP address of the cache server which is responsible to the desired We site in corresponding entry in cache routing table. For Web site community which it is responsible, the proxy stores the IP addresses of its own and next cache server, which is also responsible to the desired Web site, along the path from itself to the original Web server. The reason that we keep two separate table for information of Web sites community and cache routing is that the cache routing information don't have to be recomputed and advertised if only the contents of the Web site community are changed. Cache routing table is computed at each proxy based on the information the proxy learned either from service auto-discovery or from service advertisement. We believe an extension to the distance vector IP routing algorithm would be suitable to compute cache routing table.

6.4.3 Routing requests

To assist the cache resolution in SERA system, an efficient request routing scheme is employed. The cache resolution procedure is as follows. When a proxy receives a request for Web page from a user, it first checks its own cache for the Web page. If it has the page in cache and the page is still valid, it simply returns the page to the user. If the search returns with a miss, the proxy identifies the corresponding community ID in the Web sites community table and then looks up the cache routing table for the cache server responsible for the Web site community of the page. If it is itself the proxy responsible for the Web site, it fetches directly from the original Web site or the next cache server along the path from the itself to original Web server and cache the page. Otherwise, it forwards the request to that cache server responsible for the Web site. When a proxy receives a request for a page forwarded from another proxy, it checks its cache for the page. If it has the object in cache and it is still valid, it returns the page. If the search returns with a miss, it fetches directly from the Web server or the next cache server along the path from the itself to original Web server and cache the page.

6.5 Information dissemination

In previous approaches, information/data is disseminated via reliable multicast. However, it's well known that conventional reliable multicast protocols are too expensive which makes the entire system unscalable. Recent work done at Cornell University proposed a bimodal multicast protocol called Pbcast (Probabilistic Broadcast) [12] which scales well but also provides predictable reliability even under highly perturbed conditions (e.g. network loss rate is 20% or even 25%). In SERA system, we'll propose a piggybacked gossip protocol by integrating gossip with piggybacked scheme to further reduce the cost of gossiping. The protocol is also made to be network-aware in determining when and where to gossip. All of these will make the system scalable and fault tolerant.

6.6 Cache server cooperation

In SERA Web caching system, nearby cache servers (i.e. Web servers and proxy caches) are organized into a mesh of overlapping helper groups based on the communication latency and contents stored at cache servers. The enhanced Web performance is achieved by proper cooperation among cache servers. The cache servers inside one helper group exchange information (i.e. Web site community information, cache management information, and cache routing information) with each other via *pbcast* [12] to cooperate with each other on cache placement and replacement, prefetching, cache coherence, etc. A loose group membership is then maintained among nearby proxy caches to make the entire caching system efficient, scalable and fault tolerant. This is different from other cooperative caching schemes which use reliable multicast to distribute information.

6.7 Cache server location

The location of proxies is also important to achieve optimal Web services. In SERA, a self-organizing proxy placement algorithm is employed to adaptively add new proxies into a helper group and replicate/migrate services from the original Web server or other proxies. The objectives of such algorithm are efficient routing, efficient placement, load balancing, stable behavior, etc.

6.8 Service replication

Service replication/migration is still an open problem. One example is the Web cache placement problem. Another example is the digital library. The desirable properties of such a scheme are adaptivity, load balancing, efficient routing and efficient replication. We will propose an adaptive selective replication algorithm to resolve it.

6.9 HTTP/1.0 vs. HTTP/1.1

While both HTTP/1.0 and HTTP/1.1 are widely used, HTTP/1.1 provides much more functionality than HTTP/1.0. In SERA system, we'll take advantage of the additional features provided by HTTP/1.1. However, in absence of knowledge on how HTTP/1.1 affect Web performance, both HTTP/1.0 and HTTP/1.1 will be examined in order to gain more understanding and experiments on HTTP/1.1.

6.10 Extension to mobile environment

A natural question one may ask is that if all of the above technologies can be extended to the environment of mobile computer. What make the mobile environment different from the wired network are its inherent limitations such as the narrow bandwidth of the wireless communication channels, the relatively short active life of the power supplies (battery) of mobile units, the changing locations of required information due to client mobility, and so on. The major concern is that the clients should be able to retrieve the data of interest efficiently as well as with a minimum of energy expenditure. A widely accepted paradigm of disseminating information in a mobile environment is through the use of broadcasts. In this model the server broadcasts information and clients tune in to the broadcast to retrieve their data of interest [32]. This is based on the notion of broadcast asymmetry, where the downstream communication capacity is relatively much greater than the upstream communication capacity. However, some clients may be interested in a set of data store at server while others are interested in a different set of data. It's desirable to multicast the data to those proxies/clients which are interested in it instead of broadcast data to everybody. Prefetching and cache sharing may be very helpful to minimize the data retrieval time for clients.

7 Advantages of this approach

The SERA caching system provides us a complete solution to optimizing Web performance. Furthermore, aggregation speeds up the cache routing and facilitate connection caching. The *pbcast* is used to exchange information among nearby proxies to make the proxy communication reliable and less expensive and hence make the entire caching system scalable and fault tolerant. Network-aware prefetching minimizes the negative network effects. Cooperative piggybacked scheme assists prefetching, cache coherence, cache placement and replacement, which can further improve the performance. Adaptive service migration and self-organizing proxy location schemes facilitate dynamic data caching, load balancing, etc. Capacity planning techniques are used to solve hard decision problems in Web caching.

8 Conclusion and future work

We propose a Scalable Efficient Robust Adaptive (SERA) Web caching architecture which accommodates the exponential growth and extreme dynamic environment of the World Wide Web. In SERA caching system, Web pages are grouped by their Web sites (or subdirectories of Web site), which are further grouped into *communities* according to the servers location, user access pattern and hyperlink topology. Piggybacked prefetching/preresolving is employed based on the user access pattern and network environment information. Cooperative consistency control mechanism is employed to further improve the performance. In order to assist the cache resolution, an efficient cache routing scheme is employed. A loose group membership is maintained among nearby proxy caches to make the entire caching system scalable and fault tolerant.

This is on-going research work supervised by Srinivasan Keshav at Ensim Corp. and Balachander Krishnamurthy at AT&T Labs, we are in the stage of conducting various experiments on different data logs. Only a outline of schemes is provided in this paper. For further information, please contact either the author or the

supervisors.

References

[1] G. Abdulla, E. A. Fox, M. Abrams, and S. Williams, WWW proxy traffic characterization with application to caching (http://csgrad.cs.vt.edu/ abdulla/proxy/proxy-char.ps).

[2] M. Abrams, C. R. Standridge, G. Abdulla, S. Williams, and E. A. Fox, Caching proxies: limitations and potentials, Proceedings of the 4th International WWW Conference, Boston, MA, Dec. 1995.

[3] C. Aggarwal, J. L. Wolf, and P. S. Yu, Caching on the World Wide Web, IEEE Transactions on Knowledge and data Engineering, Vol. 11, No. 1, January/February 1999.

[4] S. Brin, Extracting patterns and relations from the World Wide Web.

[5] B. Bloom, Space/time trade-offs in hash coding with allowable errors, Communications of ACM, 13(7), pp. 422-426, July 1970.

[6] K. Bharat and A. Broder, Measuring the Web (http//www.research.digital.com/SRC/whatsnew/sem.html).

[7] P. Barford, A. Bestavros, A. Bradley, and M. E. Crovella, Changes in Web client access patterns: characteristics and caching implications, World Wide Web (special issue on Characterization and Performance Evaluation), 1999.

[8] A. Bestavros and C. Cunha, Server-initiated document dissemination for the WWW, IEEE Data Engineering Bulletin, Sept. 1996.

[9] L. Breslau, P. Cao, L. Fan, G. Phillips, S. Shenker, Web caching and Zipf-like distributions: evidence and implications, Proceedings of Infocom'99.

[10] S. Bhattacharjee, K. Calvert, and E. W. Zegura, Self-organizing wide-area network caches, IEEE Infocom'98, April 1998.

[11] J. C. Bolot and P. Hoschka, Performance engineering of the World-Wide Web: Application to dimensioning and cache design, Proceedings of the 5th International WWW Conference, Paris, France, May 1996.

[12] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu and Y. Minsky, Bimodal Multicast, Cornell CS Technical Report TR98-1683.

[13] J. W. Byers, M. Luby, M. Mitzenmacher, Accessing multiple mirror sites in parallel: using Tornado codes to speed up downloads, Proceedings of Infocom'99.

[14] V. Cate, Alex - a global file system, Proceedings of the 1992 USENIX File System Workshop, pp. 1-12, May 1992.

[15] M. Crovella and P. Batford, The network effects of prefetching, Proceedings of Infocom'98.

[16] R. Caceres, F. Douglis, A. Feldmann, G. Glass, and M. Rabinovich, Web proxy caching: the devil is in the details, ACM Performance Evaluation Review, 26(3): pp. 11-15, December 1998.

[17] A. Chankhunthod, P. B. Danzig, C. Neerdaels, M. F. Schwartz, and K. J. Worrel, A hierarchical Internet object cache, Usenix'96, January 1996.

[18] P. Cao and S. Irani, Cost-aware WWW proxy caching algorithms, Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems (USITS-97), Monterey, CA, Dec. 1997.

[19] J. Challenger, A. Iyengar, and P. Dantzig, A scalable system for consistently caching dynamic Web data, Proceedings of Infocom'99.

[20] C. R. Cunha, and C. F. B. Jaccoud, Determining WWW user's next access and its application to prefetching, Proceedings of ISCC'97: The second IEEE Symposium on Computers and Communications, July 1997.

[21] E. Cohen, B. Krishnamurthy, and J. Rexford, Improving end-to-end performance of the Web using server volumes and proxy filters, Proceedings of Sigcomm'98.

[22] E. Cohen, B. Krishnamurthy, and J. Rexford, Evaluating server-assisted cache replacement in the Web, Proceedings of the European Symposium on Algorithms-98, 1998.

[23] E. Cohen, B. Krishnamurthy, and J. Rexford, Efficient algorithms for predicting requests to Web servers, Proceedings of Infocom'99.

[24] P. Cao and C. Liu, Maintaining strong cache consistency in the World Wide Web, Proceedings of the 17th IEEE International Conference on Distributed Computing Systems, May 1997.

[25] M. C. Chan and T. Y. C. Woo, Cache-based compaction: a new technique for optimizing Web transfer, Proceedings of Infocom'99.

[26] K. Chinen and S. Yamaguchi, An interactive prefetching proxy server for improvement of WWW latency, Proceedings of INET'97, June 1997.

[27] P. Cao, J. Zhang, and K. Beach, Active cache: caching dynamic contents on the Web, Proceedings of IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing (Middleware'98), pp. 373-388.

[28] G. V. Dias, G. Cope, and R. Wijayaratne, A smart Internet caching system (http://www.isoc.org.ar/inet96/proc/a4/a4_3.htm).

[29] F. Douglis, A. Feldmann, B. Krishnamurthy, and J. Mogul, Rate of change and other metrics: a live study of the World-Wide Web, Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems (USITS-97), Dec. 1997.

[30] B. M. Duska, D. Marwood, and M. J. Feelay, The measured access characteristics of World Wide Web client proxy caches, Proceedings of USENIX Symposium on Internet Technologies and Systems (http://cs.ubc.ca/spider/feeley/wwwap/wwwap.html).

[31] A. Dingle and T. Partl, Web cache coherence, Fifth International World Wide Web Conference, Paris, France, 1996.

[32] A. Datta, D. E. VanderMeer, A. Celik and V. Kumar, Broadcast protocols to support efficient retrieval from databases by mobile users, TODS, March 1999.

[33] D. Ewing, R. Hall, and M. Schwartx, A measurement study of Internet file transfer traffic, Technical Report CU-CS-571-92, University of Colorado, Dept. of Computer Science, Boulder, Colorado, January 1992.

[34] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, Summary cache: a scalable wide-area Web cache sharing protocol, Proceedings of Sigcomm'98.

[35] A. Feldmann, R. Caceres, F. Douglis, G. Glass, and M. Rabinovich, Performance of Web proxy caching in heterogeneous bandwidth environments, Proceedings of Infocom'99.

[36] L. Fan, P. Cao, W. Lin, and Q. Jacobson, Web prefetching between low-bandwidth clients and proxies: potential and performance, Proceedings of the Sigmetrics'99.

[37] S. Galssman, A cache relay for the WWW, Proceedings of the 1st International WWW Conference, Geneva, Switzerland, May 1994 (http://www.research.digital.com/SRC/personal/Steve_Glassman/CachingTheWeb.ps).

[38] J. M. Gilbert and R. W. Brodersen, Globally progressive interactive Web delivery, Proceedings of Infocom'99.

[39] D. Gibson and J. Kleinberg, Inferring Web communities from link topology, Proceedings of the 9th ACM Conference on Hypertext and Hypermedia, 1998.

[40] D. Gibson, J. Kleinberg, and P. Raghavan, Structural analysis of the World Wide Web, WWW Consortium Web Characterization Workshop, November 1998.

[41] S. Gadde, M. Rabinovich, and J. Chase, Reduce, reuse, recycle: an approach to building large Internet caches, Proceedings of the HotOS'97 Workshop, May 1997 (http://www.cs.duke.edu/ari/cisi/crisp-recycle/crisp-recycle.htm).

[42] J. Gwetzman and M. Seltzer, The case for geographical pushing-caching, HotOS Conference, 1994 (ftp://das-ftp.harvadr.edn/techreports/tr-34-94.ps.gz).

[43] J. Gwetzman and M. Seltzer, World Wide Web cache consistency, Proceedings of the USENIX Technical Conference, pp. 141-152, January 1996.

[44] A. Heddaya, S. Mirdrad, and D. Yates, Diffusion based caching along routing paths (http://cs-www.bu.edu/faculty/heddaya/Pepers-NonTR/webcache-wkp.ps.Z).

[45] Hypertext Transfer Protocol – HTTP/1.0, RFC 1945.

[46] Hypertext Transfer Protocol – HTTP/1.1, RFC 2068.

[47] J. Jung and K. Chon, Nation-wide caching project in Korea - design and experimentation, Proceedings of the 2nd Web Cache Workshop (http://ircache.nlanr.net/Cache/Workshop97/Papers/Jaeyeon/jaeyeon.html).

[48] J. Kleinberg, Authoritative sources in a hyperlinked environment, Proceedings of the ACM-SIAM Symposium on Discrete Algorithm, 1998.

[49] M. R. Korupolu and M. Dahlin, Coordinated placement and replacement for large-scale distributed caches, Proceedings of the IEEE Workshop on Internet Applications, July 1999 (Technical Report TR-98-30, Department of Computer Science, University of Texas at Austin, December 1998).

[50] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy, Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web, STOC 1997.

[51] T. M. Kroeger, D. D. E. Long, and J. C. Mogul, Exploring the bounds of Web latency reduction from caching and prefetching, Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems, Monterey, CA, Dec. 1997.

[52] B. Krishnamurthy, J. C. Mogul, and D. M. Kristol, Key differences between HTTP/1.0 and HTTP/1.1, Proceedings of WWW-8, Toronto, May 1999.

[53] M. R. Korupolu, C. G. Plaxton and R. Rajaraman, Placement algorithms for hierarchical cooperative caching, Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms, January 1999.

[54] P. Krishnan and B. Sugla, Utility of co-operating Web proxy caches, Computer Networks and ISDN Systems, pp. 195-203, April 1998.

[55] B. Krishnamurthy and C. E. Wills, Study of piggyback cache validation for proxy caches in the World Wide Web, Proceedings of the 1997 USENIX Symposium on Internet Technology and Systems, pp. 1-12, December 1997.

[56] B. Krishnamurthy and C. E. Wills, Piggyback server invalidation for proxy cache coherency, Proceedings of the WWW-7 Conference, pp. 185-194, 1998.

[57] B. Krishnamurthy and C. E. Wills, Proxy cache coherency and replacement - towards a more complete picture, ICDC99, June 1999.

[58] I. Lovric, Internet cache protocol extension, Internet Draft <draft-lovric-icp-ext-01.txt>.

[59] A. Luotonen and K. Altis, World Wide Web proxies, Computer Networks and ISDN Systems, First International Conference on WWW, April 1994.

[60] T. S. Loon and V. Bharghavan, Alleviating the latency and bandwidth problems in WWW browsing, Proceedings of the 1997 Usenix Symposium on Internet Technologies and Systems (USITS-97), Dec. 1997.

[61] U. Legedza and J. Guttag, Using network-level support to improve cache routing, Computer Networks and ISDN Systems 30, 22-23, pp. 2193-2201, Nov. 1998.

[62] B. Li, M. J. Golin, G. F. Italiano, X. Deng, and K. Sohraby, On the optimal placement of Web proxies in the Internet, Proceedings of Infocom'99.

[63] E. Levy-Abegnoli, A. Iyengar, J. Song, and D. Dias, Design and performance of Web server accelerator, Proceedings of Infocom'99.

[64] P. Lorenzetti, L. Rizzo, and L. Vicisano, Replacement policies for a proxy cache (http://www.iet.unipi.it/luigi/research.html).

[65] I. Melve, Client-cache communication, Internet Draft <draft-melve-clientcache-com-00.txt>.

[66] E. P. Markatos and C. E. Chronaki, A TOP-10 approach to prefetching on Web, Proceedings of INET'98.

[67] A. Myers, P. Dinda, and H. Zhang, Performance characteristics of mirror servers on the Internet, Proceedings of Infocom'99.

[68] R. Malpani, J. Lorch, and D. Berger, Making World Wide Web caching servers cooperate, Proceedings of the 4th International WWW Conference, Boston, MA, Dec. 1995 (http://www.w3j.com/1/lorch.059/paper/059.html).

[69] S. Michel, K. Nguyen, A. Rosenstein, L. Zhang, S. Floyd and V. Jacobson, Adaptive Web caching: towards a new caching architecture, Computer Network and ISDN Systems, November 1998.

[70] I. Melve, L. Slettjord, T. Verschuren, and H. Bekker, Building a Web caching system - architectural considerations, Proceedings of the 8th Joint European Networking Conference, Edinburgh, Scotland, May 1997.

[71] M. Nabeshima, The Japan cache project: an experiment on domain cache, Computer Networks and ISDN System, September 1997.

[72] D. Povey and J. Harrison, A distributed Internet cache, Proceedings of the 20th Australian Computer Science Conference, Sydney, Australia, Feb. 1997.

[73] T. Palpanas and A. Mendelzon, Web prefetching using partial match prediction, Proceedings of WCW'99.

[74] V. N. Padmanabhan and J. C. Mogul, Using predictive prefetching to improve World Wide Web latency, proceedings of Sigcomm'96.

[75] M. Rabinovich, Issues in Web content replication.

[76] M. Rabinovich, J. Chase, and S. Gadde, Not all hits are created equal: cooperative proxy caching over a wide-area network, Computer Networks And ISDN Systems 30, 22-23, pp. 2253-2259, Nov. 1998.

[77] Relais: cooperative caches for the World Wide Web, 1998 (http://www-sor.inria.fr/projects/relais/).

[78] C. Roadknight and I. Marshall, Variations in cache behavior, Computer Networks and ISDN Systems, pp. 733-735, April 1998.

[79] A. Rousskov and D. Wessels, Cache Digest, Proceedings of 3rd International WWW Caching Workshop, June 1998.

[80] P. Rodriguez, C. Spanner, and E. W. Biersack, Web caching architectures: hierarchical and distributed caching, Proceedings of WCW'99.

[81] S. Sen, J. Rexford, and D. Towsley, Proxy prefix caching for multimedia stream, Infocom'99.

[82] S. Seshan, M. Stemm, and R. H. katz, SPAND: shared passive network perfoemance discovery, Proceedings of the 1st Usenix Symposium on Internet Technologies and Systems (USITS'97), December 1997.

[83] P. Scheuermann, J. Shim, and R. Vingralek, A case for delay-conscious caching of Web documents, Proceedings of the 6th International WWW Conference, Santa Clara, Apr. 1997.

[84] R. Tewari, M. Dahlin, H. Vin, and J. Kay, Beyond hierarchies: design considerations for distributed caching on the Internet, Technical Report TR98-04, Department of Computer Science, University of Texas at Austin, February 1998.

[85] R. Tewari, H. Vin, A. Dan, and D. Sitaram, Resource based caching for Web servers, Proceedings of SPIE/ACM Conference on Multimedia Computing and Networking (MMCN), January 1998.

[86] V. Valloppillil and K. W. Ross, Cache array routing protocol v1.0, Internet Draft <draft-vinod-carp-v1-03.txt>.

[87] Z. Wang, Cachemesh: a distributed cache system for World Wide Web, Web Cache Workshop, 1997.

[88] D. Wessels, Intelligent caching for World-Wide Web objects, Proceedings of INET'95, Honolulu, Hawaii, June 1995 (http://info.isoc.org/HMP/PAPER/139/archive/papers.ps.9505216).

[89] K. J. Worrell, Invalidation in large scale network object caches, M.S. Thesis, Department of Computer Science, University of Colorado, Boulder, Colorado, December 1994 (ftp://ftp.cs.colorado.edu/pub/cs/techreports/schwartz/WorrellThesis.ps.Z).

[90] R. P. Wooster and M. Abrams, Proxy caching that estimates page load delays, Proceedings of the 6th International WWW Conference, April 1997 (http://www.cs.vt.edu/ chitra/docs/www6r/).

[91] S. Williams, M. Abrams, C. R. Standridge, G. Abdulla, and E. A. Fox, Removal policies in network caches for World-Wide Web documents, Proceedings of Sigcomm'96.

[92] D. Wessels and K. Claffy, Internet cache protocol (IPC), version 2, RFC 2186.

[93] D. Wessels and K. Claffy, Application of Internet cache protocol (IPC), version 2, RFC 2187.

[94] J. Yin, L. Alvisi, M. Dahlin, and C. Lin, Using leases to support server-driven consistency in large-scale systems, Proceedings of the 18th International Conference on Distributed Computing System (ICDCS'98), May 1998.

[95] P. S. Yu and E. A. MacNair, Performance study of a collaborative method for hierarchical caching in proxy servers, Computer Networks and ISDN Systems, pp. 215-224, April 1998.

[96] Access driven Web caching, UCLA Technical Report #990007.